Optimizing Recurrent Neural Networks Architectures under Time Constraints

Junqi Jin, Ziang Yan, Kun Fu
Nan Jiang, Changshui Zhang
Dept. of Automation, Tsinghua University, Beijing, 100084, China
{jjq14, yza15, fuk11, jiangn15}@mails.tsinghua.edu.cn
zcs@mail.tsinghua.edu.cn

March 21, 2017

Abstract

Recurrent neural network (RNN)'s architecture is a key factor influencing its performance. We propose algorithms to optimize hidden sizes under running time constraint. We convert the discrete optimization into a subset selection problem. By novel transformations, the objective function becomes submodular and constraint becomes supermodular. A greedy algorithm with bounds is suggested to solve the transformed problem. And we show how transformations influence the bounds. To speed up optimization, surrogate functions are proposed which balance exploration and exploitation. Experiments show that our algorithms can find more accurate models or faster models than manually tuned state-of-the-art and random search. We also compare popular RNN architectures using our algorithms.

1 Introduction

RNNs are popular in many machine learning tasks, e.g. image caption [22][21], speech recognition [7], machine translation [1] etc. RNN's performance is highly influenced by its architecture. Researchers have devoted much effort to developing better choices, including basic units like long short-term memory (LSTM) [11] and its variants [10], gated recurrent unit (GRU) [6] and new units [12]. Besides, people research on how to connect basic units, such as [24] RNN depth definitions, [9] gated-feedback connections etc.

Comparing architectures should be fair. Fairness means same controlled conditions, e.g. same memory or number of parameters as [24][9][8][19]. Usually, people only control important conditions. [10] suggests that learning rate and hidden size are most important hyper-parameters. Even though under constrained memory of parameters, reallocating hidden sizes is still possible. Thus, it's better to find the best model after defining controlled constraint. There are many optimization techniques for RNN parameters [14] [23], however, hyper-parameters like hidden sizes are often manually tuned.

This paper mainly considers choices of layers' hidden sizes which are discrete hyper-parameters under some constraints. There are several algorithms for tuning hyper-parameters. [18] propose a gradient method, however, only suitable for continuous hyper-parameters. General frameworks like Bayesian optimization [20][5] and random search [3] do not utilize prior knowledge about hidden sizes.

We propose novel algorithms to optimize the model by choosing proper hidden sizes under some constraints. The algorithm applies to many constraints such as memory or number of parameters, power of model etc. We choose running time as constraints, because many applications care about the best performance under constrained time. Our contributions are as follows: (1)We convert this discrete optimization into a subset selection problem; (2)Transformation is proposed to make objective function submodular and constraint function supermodular; (3)We propose a greedy algorithm to solve the problem with theoretical bounds and point out relationship between the transformation and bounds; (4)Surrogate function is proposed to speed up the optimization; (5)Having considered other hyper-parameters, we evaluate our algorithms on text8 and WMT data set. Different basic units and different connections are compared.

Results show our algorithm can find more accurate model or faster model than manually tuned state-of-the-art and random search.

2 Approach

2.1 Optimization of Architectures

Search of Architecture Space Neuron number of a layer is called 'width' denoted as w. All input, hidden layers' widths are denoted as $[w_1, w_2...w_k] = \phi$. With specific connections, once ϕ is determined, training and validation give validated performance P and running time T (not training time). The search of architecture space formed by all ϕ is defined as Eq. 1 where B is time budget.

$$\max_{\phi} \ P(\phi) \quad s.t. \ T(\phi) \leq B \tag{1}$$

Subset Selection Problem We propose a method to convert each ϕ to a set S. Suppose an RNN's $\phi = [w_1, w_2...w_k]$, we define value range of w_i as $[0, W_i]$. For ϕ's i_{th} dimension, we construct set $V^i = \{u_1^i, u_2^i...u_{W_i}^i\}$, where elements $u_j^i, j = 1...W_i$ have same effects to ϕ, each of which, if selected, contributes one increased neuron. Any $w_i < W_i$ corresponds $S^i = \{u_1^i, u_2^i...u_{w_i}^i\} \subset V^i$. Then ϕ corresponds to set $S = \cup_{i=1}^k S^i$ where $\forall p \neq q, S^p \cap S^q = \emptyset$ because u^p, u^q are from different layers. The universal set is $V = \cup_{i=1}^k V^i$. Based on this conversion, if $S_A \subset S_B$, then $\forall i, S_A^i \subset S_B^i$, thus $w_{Ai} \leq w_{Bi}$. Another property is that any ϕ is able to be converted to a set S, which means the selection of subset of V is equivalent to search of ϕ space. And problem Eq. 1 is converted to Eq. 2.

$$\max_{S} \ P(S) \quad s.t. \ T(S) \leq B \tag{2}$$

Usually w_i's value range is $[W_{low,i}, W_{high,i}]$. Define $\phi_0 = [W_{low,1}, W_{low,2}...W_{low,k}]$, and offset ϕ as $\phi_{off} = \phi - \phi_0$, then we convert sets using ϕ_{off}. Further, ϕ_0 corresponds to $S = \emptyset$ and $[W_{high,1}, W_{high,2}...W_{high,k}]$ corresponds to V.

2.2 Properties of P(S) and T(S)

2.2.1 Monotonically Increasing

Consider $S_A \subset S_B$, from section 2.1 $w_{Ai} \leq w_{Bi}$. Suppose S_A's RNN has parameter $W_A \in \mathbb{R}^{m \times n}$, and corresponding parameter for S_B is $W_B \in \mathbb{R}^{M \times N}$ where $m \leq M, n \leq N$ (because $w_{Ai} \leq w_{Bi}$). For all parameters, by setting W_B top-left $m \times n$ sub-matrix as W_A and rest elements in W_B zeros, we prove S_A and S_B's RNNs have identical computation process and $P(S_A) = P(S_B)$ holds true for architectures involved in this paper. (proofs in supplementary) Thus, we train an RNN S_{new} with random initial parameters, and based on validation history, if there exists $S_{small} \subset S_{new}$ and $P(S_{small}) \geq P(S_{new})$, we transfer parameters from S_{small} to S_{new} making $P(S_{new}) = P(S_{small})$. Then extremely slightly training of S_{new} using gradients from validation set makes $P(S_{new})$ extremely slightly larger than $P(S_{small})$. (Validation set should not be used to compute gradients, however, we only apply this method when $P(S_{new}) = P(S_{small})$, and $P(S_{new})$ is not final output since S_{small} performs nearly the same with shorter time). Thus, for any $S_A \subset S_B$, we guarantee $P(S_A) < P(S_B)$. For running time, larger parameter matrices cost more computation time, so $T(S_A) < T(S_B)$. Thus, $P(S), T(S)$ are both monotonically increasing. We revise $P(S) := P(S) - P(\emptyset), T(S) := T(S) - T(\emptyset)$ to make revised $P(\emptyset) = T(\emptyset) = 0$.

2.2.2 Submodular and Supermodular

Interpolation Function $p(\cdot)$ Since $P(\cdot)$ is monotone, monotone spline interpolation for $P(\cdot)$ gives continuous function $p(\cdot)$ satisfying: (1) for any $\phi \in \mathbb{Z}^k$, $P(\phi) = p(\phi)$; (2) monotone property is maintained (derivatives $p_{w_i} > 0, i = 1...k$); (3) second order derivatives of $p(\cdot)$ exist.

Conditions for Submodularity Consider changing only two different layers in $p(\cdot)$ denoted as $p(x, y)$, and there's Thrm. 1(proofs in supplementary). The intuition is: marginal gain $P(u|S) = P(S \cup \{u\}) - P(S)$ corresponds to derivative p_x. $p_{xx} \leq 0$ means p_x decreases as x increases, corresponding to $P(u|S)$ diminishing as the layer u belonging to increases width. $p_{xy} \leq 0$ means $P(u|S)$ diminishes as the layer u not belonging to increases width. Consider $S_A \subset S_B$, thus $w_{Ai} \leq w_{Bi}$, imagine S_A grows to S_B by adding neurons one by one to each layer. No matter which layer is added, all layers' marginal gain diminish, therefore, at last $P(u|S_A) \geq P(u|S_B)$, which means $P(S)$ is submodular. Supermodularity ($P(u|S_A) \leq P(u|S_B)$) is concluded similarly.

Theorem 1. *If all two-layer pairs' second order derivatives satisfy $p_{xx} \leq 0, p_{yy} \leq 0, p_{xy} \leq 0$ ($p_{xx} \geq 0, p_{yy} \geq 0, p_{xy} \geq 0$), set function $P(S)$ is submodular (supermodular).*

Saturated and Explosive Transformation $p(x, y)$ may not satisfy Thrm 1 naturally. Thus, we convert Eq. 2 to $\max_S P_{\mu_1}(S) \;\; s.t. \; T_{\mu_2}(S) \leq B_{\mu_2}$ where $P_{\mu_1}(S) = \gamma_1(\mu_1 P(S)), T_{\mu_2}(S) = \gamma_2(\mu_2 T(S)), B_{\mu_2} = \gamma_2(\mu_2 B)$, γ_1, γ_2 are designed monotone increasing functions and μ_1, μ_2 are positive numbers. This new problem is equivalent to Eq. 2 because an inequality multiplied by a positive number and applied with monotone increasing function composition is equivalent to original inequality. Appropriate γ, μ make $P_{\mu_1}(S)$ submodular and $T_{\mu_2}(S)$ supermodular. The second order derivatives for transformed interpolation function $p_{\mu_1} = \gamma_1(\mu_1 p(x, y))$ is as Eq. 3.

$$p_{\mu_1,xx} = \frac{\partial^2 \gamma_1(\mu_1 p(x,y))}{\partial^2 x} = \gamma_1''(\mu_1 p) p_x^2 \cdot \mu_1^2 + \gamma_1'(\mu_1 p) p_{xx} \cdot \mu_1 \tag{3}$$

Setting $p_{\mu_1,xx} \leq 0$, choosing $\gamma'' < 0$, (a good choice is $\gamma_1(z) = 1 - exp(-z)$ making $\gamma_1'/\gamma_1'' = -1$) Eq. 3 concludes $\forall x, \mu_1 \geq \frac{p_{xx}}{p_x^2}(\frac{\gamma_1'}{-\gamma_1''}) = \frac{p_{xx}}{p_x^2}$.

More Saturated Transformation Setting all two-layer pairs' $p_{\mu_1,xx} \leq 0, p_{\mu_1,yy} \leq 0, p_{\mu_1,xy} \leq 0$, choosing $\gamma_1(z) = 1 - exp(-z)$, knowing $p_x > 0, p_y > 0$, there is condition Eq. 4. If μ_1 is chosen according to Eq. 4, based on Theorem 1, $P_{\mu_1}(S)$ is submodular. Note if all $p_{xx} \leq 0, p_{xy} \leq 0, p_{yy} \leq 0$ are already true, original $P(S)$ is submodular, such transformation is not needed.

$$\mu_1 \geq \mu_{1ms} = \max_{all\ two-layer\ pairs} \max_{x,y} \left(\frac{p_{xx}}{p_x^2}, \frac{p_{xy}}{p_x p_y}, \frac{p_{yy}}{p_y^2}\right) \tag{4}$$

Less Saturated Transformation If $p_{xx} \leq 0, p_{xy} \leq 0, p_{yy} \leq 0$ are already true, setting all $p_{\mu_1,xx} \leq 0, p_{\mu_1,yy} \leq 0, p_{\mu_1,xy} \leq 0$, $\gamma_1(z) = exp(z) - 1$, there's condition Eq. 5. $P_{\mu_1}(S)$ is still submodular if μ_1 is chosen according to Eq. 5. We show its usage in section 2.3.

$$\mu_1 \leq \mu_{1ls} = - \max_{all\ two-layer\ pairs} \max_{x,y} \left(\frac{p_{xx}}{p_x^2}, \frac{p_{xy}}{p_x p_y}, \frac{p_{yy}}{p_y^2}\right) \tag{5}$$

Similarly, $T(S)$ has more explosive transformation $\mu_2 \geq \mu_{2me}$ and less explosive transformation $\mu_2 \leq \mu_{2le}$. (see supplementary) μ_1, μ_2 control transformation extent, when μ_1, μ_2 are near zeros, according to Taylor expansion, it becomes linear transformation.

Estimation of Critical μ Using validation history $\{\phi_i, P(\phi_i)\}_{i=1}^N$, we regress $P(S)$ by locally weighted quadratic regression $\hat{p}(\phi) = \sum_{h=1}^k \sum_{j=1}^h \alpha_{hj} w_h w_j + \sum_{j=1}^k \beta_j w_j + \beta_0$, where α, β are parameters. Minimizing $cost = \sum_{i=1}^N \delta_i (\hat{p}(\phi_i) - P(\phi_i))^2$ is solved through normal equation, (see supplementary) where $\delta_i = exp(-\frac{\|\phi - \phi_i\|_2^2}{2\tau^2})$ controls cost weights for a test point ϕ, τ is chosen through leave-one-out validation. The estimated $\hat{p}_{w_i} = 2\alpha_{ii} w_i + \sum_{h=i+1}^k \alpha_{hi} w_h + \sum_{j=1}^{i-1} \alpha_{ij} w_j + \beta_i$, and $\hat{p}_{w_i w_j} = \alpha_{ji}$ if $j > i$, $\hat{p}_{w_i w_j} = \alpha_{ij}$ if $j < i$, $\hat{p}_{w_i w_i} = 2\alpha_{ii}$. For $P(S), T(S)$, gridded points' derivatives in ϕ space are estimated, which are used for estimation of $\mu_{1ms}, \mu_{1ls}, \mu_{2me}, \mu_{2le}$.

2.3 Greedy Algorithms

Section 2.2 concludes submodular $P_{\mu_1}(S)$ and supermodular $T_{\mu_2}(S)$. There are many submodular maximization algorithms [15] [17]. Different from most constraints, $T_{\mu_2}(S)$ is a supermodular function. Thus, we propose Alg. 1 to solve the optimization.

Firstly setting $\delta(S,X) = P_{\mu_1}(S \cup X) - P_{\mu_1}(S)$, Alg. 1 gives solution S_u, then setting $\delta(S,X) = \frac{P_{\mu_1}(S\cup X)-P_{\mu_1}(S)}{T_{\mu_2}(S\cup X)-T_{\mu_2}(S)}$, Alg. 1 gives solution S_s. $S^* = argmax_{S\in\{S_u,S_s\}} P_{\mu_1}(S)$ is final solution. Borrowing ideas from [13] [16], we prove S^* satisfies Thrm. 2 and Thrm. 3. (proofs in supplementary) We introduce $\kappa_T = \min_{j\in V} \frac{T(j)}{T(V)-T(V\setminus j)}$ as curvature of $T(S)$. If $T(S)$ is supermodular, $0 \le \kappa_T \le 1$, and curvature measures the distance of $T(S)$ from modularity. $\kappa_T = 1$ if and only if $T(S)$ is modular, or $T(S) = \sum_{j\in S} T(j)$.

1: Initialize: $S \leftarrow \emptyset, U = V$
2: **while** $U \neq \emptyset$ **do**
3: **for each** $X \in U$, compute $\delta_X = \delta(S,X)$
4: $X^* = argmax\{\delta_X : X \in U\}$
5: **if** $T_{\mu_2}(S \cup X^*) \le B_{\mu_2}$, **then** $S = S \cup X^*$
6: $U = U \setminus X^*$
7: **end while**
8: **return** S

Algorithm 1: Greedy Algorithm

Theorem 2. *If S^* is the solution from Algorithm 1 when $T_{\mu_2}(S) \le B_{\mu_2}$. Then, $P_{\mu_1}(S^*) \ge \frac{1}{2}(1-\frac{1}{e})P_{\mu_1}(S^{opt}_{small})$, where S^{opt}_{small} is the global optimum when $T_{\mu_2}(S) \le \kappa_{T_{\mu_2}} B_{\mu_2}$.*

In Thrm.2, since $0 \le \kappa_{T_{\mu_2}} \le 1$, $\kappa_{T_{\mu_2}} B_{\mu_2} \le B_{\mu_2}$. Thrm.2 gives a $P_{\mu_1}(S^*)$'s lower bound based on the optimum of a smaller budget problem.

Theorem 3. *S^* from Algorithm 1 when $T_{\mu_2}(S) \le B_{\mu_2}$ satisfies $P_{\mu_1}(S^*) \ge \frac{1}{m+1}(1-\frac{1}{e})P_{\mu_1}(S^{opt})$, where S^{opt} is the global optimum when $T_{\mu_2}(S) \le B_{\mu_2}$ and $m = max\{|S| : T_{\mu_2}(S) \le \frac{B_{\mu_2}}{\kappa_{T_{\mu_2}}}\}$.*

Thrm.3 gives a lower bound based on the same budget B_{μ_2} problem, however, with a smaller approximation factor. m is determined by the curvature $\kappa_{T_{\mu_2}}$. We also proved that as $\kappa_{T_{\mu_2}}$ becomes large, m becomes small, and in modular case $\kappa_{T_{\mu_2}} = 1$, the bound becomes $P_{\mu_1}(S^*) \ge \frac{1}{2}(1-\frac{1}{e})P_{\mu_1}(S^{opt})$. Note $P(S)$ is revised by subtracting the smallest model's performance, e.g. the smallest model ϕ_0 has accuracy 0.91, and best accuracy is 1.00, this bound means $P(S) = accuracy(S) - 0.91 > \frac{1}{2}(1-\frac{1}{e})(1.00-0.91)$. Thrm.2 and 3 are extensions of [16]'s modular case which is a special case if Thrm.2 and 3 use $\kappa_{T_{\mu_2}} = 1$.

Transformation of P(S) and T(S) We actually care about $P(S^*)$ instead of $P_{\mu_1}(S^*)$. Thrm.2 and 3 give bound $\gamma_1(\mu_1 P(S)) \ge a\gamma_1(\mu_1 P(S^{opt}))$ where a is approximation factor. Thus, $P(S) \ge \frac{1}{\mu_1}\gamma_1^{-1}(a\gamma_1(\mu_1 P(S^{opt}))) = L(\mu_1)$. We proved: (proofs in supplementary) if $P(S)$ is not submodular and $\gamma_1(z) = 1-exp(-z)$, $L(\mu_1)$ is monotone decreasing and $\lim_{\mu_1\to 0} L(\mu_1) = aP(S^{opt})$, $\lim_{\mu_1\to\infty} L(\mu_1) = 0$, which means μ_{1ms} is the best μ_1; if $P(S)$ is submodular and $\gamma_1(z) = exp(z)-1$, $L(\mu_1)$ is monotone increasing and $\lim_{\mu_1\to 0} L(\mu_1) = aP(S^{opt})$, $\lim_{\mu_1\to\infty} L(\mu_1) = P(S^{opt})$, which means μ_{1ls} is the best μ_1. Denote $\kappa(\mu_2) = \kappa_{T_{\mu_2}} = \frac{\gamma_2(\mu_2 T(j))}{\gamma_2(\mu_2 T(V))-\gamma_2(\mu_2 T(V\setminus j))}$, we proved: if $T(S)$ is not supermodular and $\gamma_2(z) = exp(z)-1$, $\kappa(\mu_2)$ is monotone decreasing and $\lim_{\mu_2\to 0}\kappa(\mu_2) = \kappa_T$, $\lim_{\mu_2\to\infty}\kappa(\mu_2) = 0$, which means μ_{2me} is the best μ_2; if $T(S)$ is supermodular and $\gamma_2(z) = 1-exp(-z)$, $\kappa(\mu_2)$ is monotone increasing and $\lim_{\mu_2\to 0}\kappa(\mu_2) = \kappa_T$, $\lim_{\mu_2\to\infty}\kappa(\mu_2) = \infty$, which means μ_{2le} is the best μ_2. Therefore, $\mu_{1ms}, \mu_{1ls}, \mu_{2me}, \mu_{2le}$ give tightest bounds for $P(S)$ (larger $\kappa_{T_{\mu_2}}$ and higher $L(\mu_1)$).

2.4 Surrogate Functions

Frequent accessing P, T in Alg. 1 is expensive, so surrogate functions speed up the optimization.

2.4.1 Bounds of Unevaluated Points

We propose bounds for unevaluated points. Consider monotone increasing submodular function $f(S)$ and S_A, S_B whose architectures are $\phi_A = [a_1, a_2...a_d], \phi_B = [b_1, b_2...b_d]$. Denote $S_B \setminus S_A = \{e_1, e_2...e_m\}, S_A \setminus S_B = \{e'_1, e'_2...e'_n\}$ and $S_i = S_A \cup \{e_1, e_2...e_i\}, S_0 = S_A, S'_i = S_B \cup \{e'_1, e'_2...e'_i\}, S'_0 = S_B$. $f(S_A \cup (S_B \setminus S_A)) = f(S_A \cup S_B) = f(S_B \cup (S_A \setminus S_B))$ gives Eq. 6.

$$f(S_A) + \sum_{i=1}^{m} f(e_i|S_{i-1}) = f(S_A \cup S_B) = f(S_B) + \sum_{i=1}^{n} f(e'_i|S'_{i-1}) \tag{6}$$

In Eq. 6, since $f(S)$ is submodular, $\sum_{i=1}^{m} f(e_i|S_A) \geq \sum_{i=1}^{m} f(e_i|S_{i-1})$. And $f(S)$ is monotone increasing means $\sum_{i=1}^{n} f(e'_i|S'_{i-1}) \geq 0$. Therefore, based on Eq. 6 there is Eq. 7.

$$f(S_A) + \sum_{i=1}^{m} f(e_i|S_A) \geq f(S_B) \tag{7}$$

Any $e_i \in S_B \; and \; e_i \notin S_A$ comes from ϕ's dimension j that $b_j > a_j$, e.g. elements from dimension j in S_A is $\{u_1^j, u_2^j ... u_{a_j}^j\}$, and S_B's is $\{u_1^j, u_2^j ... u_{a_j}^j, u_{a_j+1}^j ... u_{b_j}^j\}$. Therefore, $\sum_{i=1}^{m} f(e_i|S_A) = \sum_{j:b_j>a_j} \sum_{k=a_j+1}^{b_j} f(u_k^j|S_A)$. Because in the same dimension j, elements have same effects to architecture (same layer and one neuron increased), we denote them as u^j and get $\sum_{i=1}^{m} f(e_i|S_A) = \sum_{j:b_j>a_j} (b_j - a_j) f(u^j|S_A) = \sum_{j=1}^{d} max(b_j - a_j, 0) f(u^j|S_A)$. Thus Eq. 7 becomes Eq. 8.

$$\sum_{j=1}^{d} max(b_j - a_j, 0) f(u^j|S_A) \geq f(S_B) - f(S_A) \tag{8}$$

There's another inequality. If $S_A \subset S_B$, Eq. 6 becomes $f(S_A) + \sum_{i=1}^{m} f(e_i|S_{i-1}) = f(S_B)$. Submodularity gives $f(S_A) + \sum_{i=1}^{m} f(e_i|S_B) \leq f(S_B)$. Converting e_i to u^j concludes Eq. 9.

$$\sum_{j=1}^{d} max(b_j - a_j, 0) f(u^j|S_B) \leq f(S_B) - f(S_A) \tag{9}$$

If S_A, S_B are evaluated, we propose following bounds for unevaluated $S_C \; with \; \phi_C = [c_1, c_2...c_d]$.

Monotone Upper Bounds If $S_C \subset S_A$, there is $f(S_C) \leq f(S_A)$.

First Submodular Upper Bound If $S_B \setminus S_A \neq \emptyset \; and \; S_C \subset S_A$, apply Eq. 8 to S_A, S_B and apply Eq. 9 to S_C, S_A, there are Eq. 10 11.

$$\sum_{j=1}^{d} max(b_j - a_j, 0) f(u^j|S_A) \geq f(S_B) - f(S_A) \tag{10}$$

$$\sum_{j=1}^{d} max(a_j - c_j, 0) f(u^j|S_A) \leq f(S_A) - f(S_C) \tag{11}$$

Define α as Eq. 12, where $\epsilon > 0$ is very small. Eq. 10 11 conclude bound Eq. 12.

$$\alpha = \min_j \frac{max(a_j - c_j, 0)}{max(b_j - a_j, 0) + \epsilon}, \quad f(S_C) \leq f(S_A) - \alpha(f(S_B) - f(S_A)) \tag{12}$$

Second Submodular Upper Bound If $S_C \setminus S_A \neq \emptyset \; and \; S_B \subset S_A$, another bound is Eq. 13.

$$\alpha = \min_j \frac{max(a_j - b_j, 0)}{max(c_j - a_j, 0) + \epsilon}, \quad f(S_C) \leq f(S_A) + \frac{1}{\alpha}(f(S_A) - f(S_B)) \tag{13}$$

Our method uses evaluated points to estimate the increasing rate and bound unevaluated points based on diminishing return property. For an unevaluated point S_C, all possible pairs of evaluated points are used to compute bounds, and minimum of all upper bounds is $f(S_C)$'s final upper bound $f_{upper}(S_C)$. Similarly there's lower bound $g_{lower}(S_C)$ for supermodular function $g(S)$. (see supplementary)

2.4.2 Balance Exploration and Exploitation

We compute $P_{upper}(S)$ for $P_{\mu_1}(S)$, $T_{lower}(S)$ for $T_{\mu_2}(S)$. $P_{upper}(S), T_{lower}(S)$ are also monotone. (proofs in supplementary) Thus, transforming again makes $P_{upper,\mu_1}(S)$ submodular and $T_{lower,\mu_2}(S)$ supermodular which are used as surrogate functions. Surrogate functions have these properties: evaluated points are true values; unevaluated points' P is overrated and T is underrated because of upper and lower bounds. Using surrogate functions, in Alg. 1, unevaluated points have higher δ_X than it should be, leading to natural exploration mechanism.

The whole procedure: **step 1**, evaluate several random points as evaluation history; **step 2**, convert $P(S), T(S)$ to $P_{\mu_1}(S), T_{\mu_2}(S)$; **step 3**, compute $P_{\mu_1}(S), T_{\mu_2}(S)$'s bounds; **step 4**, transform bounds to obtain surrogate functions; **step 5**, Alg. 1 using surrogate functions proposes a new candidate S^* and intermediate used S during $\emptyset$ grows to S^*; **step 6**, evaluate S^* and randomly selected intermediate S, and add results to evaluation history; go back to **step 2**. Repeat until S^* is stable.

In step 5, Alg. 1 chooses set S growing path by comparing evaluated points' true marginal gains and unevaluated overrated marginal gains. If unevaluated points win, step 6 evaluates this growing path and their marginal gains drop to true values, which represents exploration. If evaluated points win, more exploration only gives worse paths, thus, current path is stable. A typical path see Fig. 2(a).

2.5 Recurrent Neural Networks Architectures

We compare basic units (LSTM, GRU), each being a circle node in Fig. 1. Secondly, connection methods, according to recent suggestions [9][24][19], are evaluated, including forward depth (f) meaning f transformations (vertical arrows in Fig. 1(a)) from input to output, recurrent depth (r) meaning r transformations (horizontal arrows in Fig. 1(a)) between adjacent time steps, skip coefficients (s) meaning time step t receives shortcut transformations from time step $t - s$. Skip coefficient's evaluation includes bottom-bottom (bb), top-top (tt), bottom-up (bu) and top-down (td) as Fig. 1(b). Thirdly, attention mechanism, a special RNN architecture is evaluated, which has been broadly adopted in image caption [22], speech recognition [7] and machine translation [1]. (details in supplementary)

Above architectures are optimized under validation time constraints. For architectures in Fig. 1(a)(b), widths of input feature x and basic units' hidden layer $h_1, h_2, ...$ as $[w_x, w_{h_1}, w_{h_2}...]$ are optimized. For attention structure, widths of input feature, encoder RNN hidden layer, attention hidden layer, decoder RNN hidden layer and output feature as $[w_x, w_e, w_a, w_d, w_y]$ are optimized as Fig. 1(c).

3 Experiments

Dataset Following [24], text8 is used for character level language modeling, which has 100M characters. Cost uses $-log_2$ of perplexity, and 10.0 minus cost is performance (better model has higher performance). Each width's range is [3, 500]. Attention architecture is evaluated on English-French translation problem. We use bilingual parallel corpora news-commentary (5.5M words) which is part of WMT-14 data. Following [1], we use Moses for tokenization and top frequent 5000 words in each language as vocabulary which covers 90 percent of all words. Any word not included is mapped to UNK token. Sentences with length up to 30 words are used. BLEU-1 score (without UNK) from Moses is evaluation criterion. Each width's range is [3, 200].

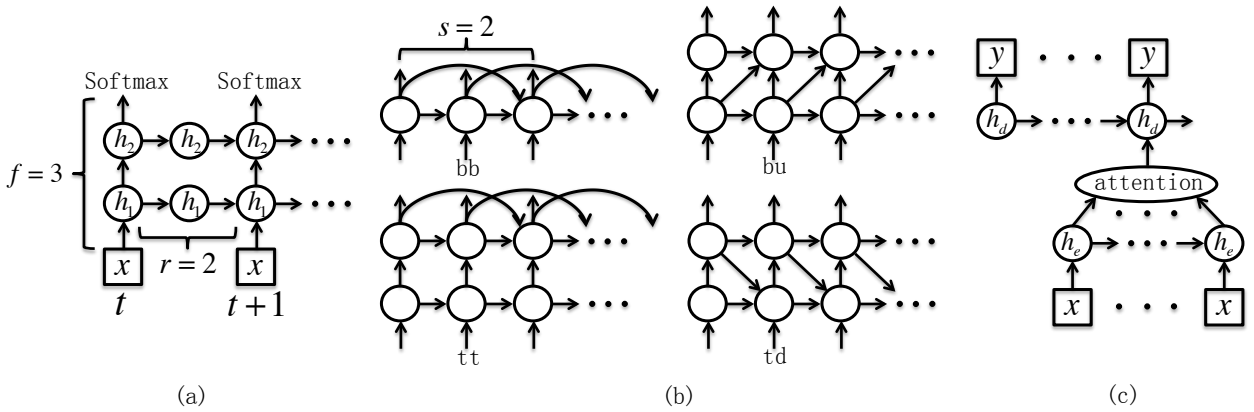

Figure 1: (a)forward, recurrent depth; (b)skip coefficient(s_{tt}=s_{bb}=2,s_{bu}=s_{td}=1); (c)attention architecture

Other Hyper-parameters For all experiments, we use Adam [14] for optimization with $\beta_1 = 0.9, \beta_2 = 0.999$. For each evaluation, grid search is used for learning rate in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. We initialize hidden to hidden matrices with random orthogonal matrices and other parameters with uniform distribution from $[-10^{-3}, 10^{-3}]$. LSTM's forget gates are initialized with 2. Both data sets are split into training, validation and test sets with ratio $0.90 : 0.05 : 0.05$. Training mini-batch size is 50. Early stopping by monitoring validation cost is used. Training is based on GPU using Theano [4][2]. Validation running time is based on CPU processor clock describing time needed for one RNN's step averagely. Program see: https://github.com/jinjunqi/monogreedy

3.1 Critical μ for Transformations

By estimating critical μ, we analyze properties of $P(S), T(S)$. To conduct more experiments, we use a smaller set, 2M of text8 divided into non-overlapping sequences with length 20. For each model in Fig. 1(a)(b), we estimate derivatives using about 40 evaluated random points. (Typical derivatives as Fig. 2) Interestingly, all models satisfy $p_{xy} \leq 0, t_{xy} \geq 0$, which means original $P(S)$ is submodular and $T(S)$ is supermodular. Some μ for less saturated and less explosive transformation are as Tab. 1. $P(S)$ saturates fast, and μ_{ls} is not small. $T(S)$ is nearly modular when model is small, leading to very small μ_{le}, and supermodularity becomes obvious when all widths grow large.

Ignoring formulation, we use less saturated transformation μ_{ls} from 1e3 to 1e-3 and more saturated transformation μ_{ms} from 1e-3 to 1e3 for $P(S)$ in section 2.4.2's **step 2** to check how optimization is influenced by μ. r=1, f=2 LSTM's result is as Fig. 3(f)top (μ=1e-3 means nearly linear transformations, thus, are merged). As $P(S)$ is already submodular, it seems more saturated transformation hurts the performance. Large range around the critical μ (marked as blue 'X' in Fig. 3(f)top) performs best. Less saturated transformation with too large μ leads to slightly worse performance with which $P(S)$ is no longer submodular. Other models show similar results. Surrogate functions become more accurate as

Table 1: 2M of text8, LSTM, μ for $P(S), T(S)$

		r=1	r=2	r=3		bb	tt	bu	td
$\mu_{ls}\ of\ P$	f=2	1.21	2.50	0.0358	s=3	0.1592	0.4803	0.3406	0.0096
$\mu_{le}\ of\ T$	f=2	3.99e-5	1.67e-5	6.91e-5	s=3	6.81e-6	4.51e-7	8.19e-6	1.20e-3

more points are evaluated. Fig. 3(f)bottom compare root square error of monotone and submodular bounds on hold-out validation evaluation history, showing submodular bound's benefit.

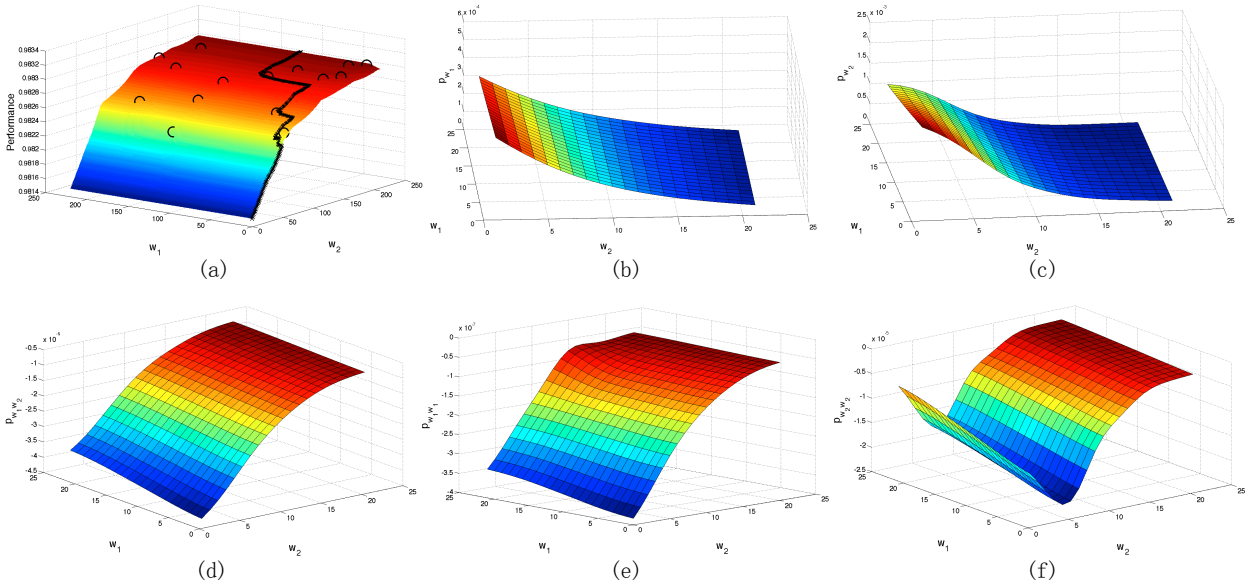

Figure 2: $r = 1, f = 2$ LSTM: (a)Surrogate function for $P(S)$, performance vs $[w_1, w_2]$. 'O' are evaluated points, 'X' are greedy selection path. (b)p_{w_1} (c)p_{w_2} (d)$p_{w_1 w_2}$ (e)$p_{w_1 w_1}$ (f)$p_{w_2 w_2}$ (grid step=10)

3.2 Compare Basic Units and Connections

Using 2M of text8 data, (LSTM, GRU) with (r=1,2,3; f=2,3) and (bb, tt, bu, td using s=1,3,5) architectures are compared. We set validation time constraints from 0 to 300 at step 20 (e-5 seconds), and at each time constraint, our algorithms optimize widths to give corresponding best architecture whose performance and time plotted in Fig. 3. Evaluating one point takes about 40-60 minutes. Fig. 3(a)(b) show that larger s is not helpful, and bu, td perform best. Fig. 3(c) tells that larger r hurts performance, and (r=1, f=3) works best. Each figure's winner is plotted in Fig. 3(d), showing that LSTM is better than GRU, and (bu, s=1) is best. Tab. 2 gives results when time constraint is 300. Checking relationship of different layers' widths of best architectures, we find as a model grows large, layers' widths are generally positively correlated. Following [24], using 100M text8 data divided into 180 length sequences, choosing same LSTM (td, s=1),

Table 2: 2M of text8, $T(S) < 300$e-5 seconds, architectures' $-log_2(perplexity)$

		f=2	f=3		bb	tt	bu	td
LSTM	r=1	1.8230	**1.8136**	s=1	1.8230	1.8136	**1.8024**	1.8144
	r=2	1.8261	1.8316	s=3	1.8839	1.8259	1.8341	1.8375
	r=3	1.8458	1.9117	s=5	1.9249	1.8404	1.8671	1.8519
GRU	r=1	1.8578	**1.8446**	s=1	1.8578	1.8446	1.8489	**1.8419**
	r=2	1.8533	1.8481	s=3	1.9458	1.8735	1.8541	1.8837
	r=3	1.8622	1.8606	s=5	1.9825	1.9005	1.8899	1.8880

through setting different validation time constraints, our algorithms can find faster models with same performance, and more accurate models with same running time than state-of-the-art. Evaluating each point takes about 1 day. Results on test set see Tab. 3.

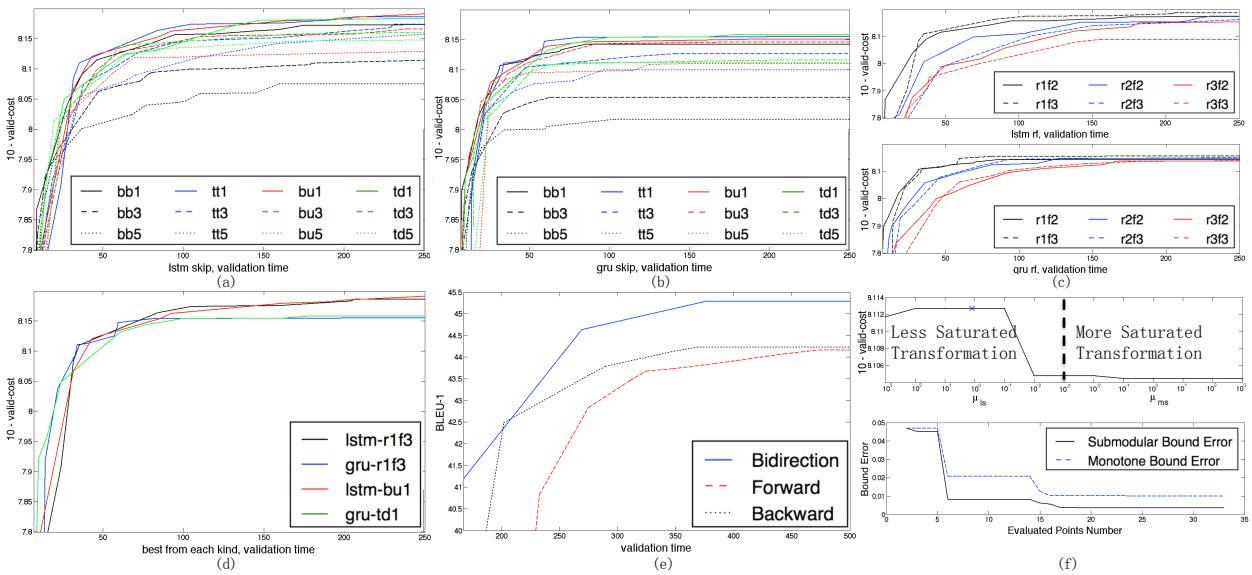

Figure 3: Best architectures' performance vs running time: (a)LSTM (s=1,3,5; bb,tt,bu,td) (b)GRU (s=1,3,5; bb,tt,bu,td) (c)(r=1,2,3; f=2,3), (top: LSTM), (bottom: GRU) (d)winners of (a)-(c) (e)attention architecture (f)top: performance vs extent of transformation; bottom: bound error vs evaluated points number.

Table 3: 100M of text8, 10 trials, time (e-5 seconds)

	$[w_x, w_{h1}, w_{h2}]$	test time	test $-log_2(perplexity)$
[24]	[256, 256, 256]	144.5	1.63
Faster models	[334,122,332]±[52,2,6]	**106**±**4**	1.6288±0.0005
More accurate models	[332,176,358]±[52,8,4]	143.8±0.2	**1.569**±**0.006**

3.3 Attention Architecture

Attention architecture also needs less saturated, less explosive transformations. Setting time constraints 100 to 500 at step 50 e-5 seconds, we optimize bidirectional [1] and one directional (only forward or backward feed source sentences) models. Validation results in Fig. 3(e) show bidirection is the best and backward feeding is slightly better than forward. Evaluating each point takes about 6 hours. Under same constraint, our algorithm is compared with random search [3] which uses same number of evaluated points as ours to optimize models. Random range uses widths' range and points exceeding constraint are rejected. Test results in Tab. 4 show that both algorithms give similar shape architectures. Ours achieves better BLEU-1, and gives finer results: closer time to constraint and lower deviations of widths, time and BLEU-1.

Table 4: part of WMT, $T(S)$<380, time (e-5 seconds), 5 trials, BLEU-1 (higher is better)

	$[w_x, w_e, w_a, w_d, w_y]$	test time	test BLEU-1
Random search	[64,54,87,38,16]±[12,8,14,13,13]	374±4	44.4±0.8
Ours	[51,46,137,31,11]±[4,5,28,5,2]	**379.1**±**0.3**	**45.29**±**0.02**

4 Discussion

We converted RNN's hidden size optimization to a subset selection problem, and proposed function transformation to make objective function submodular and constraint function supermodular. We designed greedy algorithms to solve the problem. Experiments showed our algorithm's effectiveness in finding more accurate model and faster model than manually tuned state-of-the-art and random search algorithm. We chose time as constraint, however, our algorithm is suitable for many other monotone constraints. Our experiments used text8 and part of WMT data, results showed this paper's RNN's performance were submodular and running time supermodular. In text8 task, we found recurrent depth and skip coefficient were not helpful, td, bu models performed best, and LSTM was better than GRU. In WMT task, we found bidirectional encoder was better than one-directional method. All results were based on two specific data set, and it's possible to obtain different results on other tasks. In future, we'll try more architectures, tasks and constraints using our algorithms.

Acknowledgement

This work is supported by 973 Program(2013CB329503)

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *Proceedings of the International Conference on Learning Representations*, 2015.

[2] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.

[5] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[6] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014.

[7] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[9] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2067–2075, 2015.

[10] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.

[13] Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*, 2015.

[15] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3:19, 2012.

[16] Andreas Krause and Carlos Guestrin. A note on the budgeted maximization of submodular functions. 2005.

[17] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[18] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2113–2122, 2015.

[19] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *Proceedings of the International Conference on Learning Representations*, 2014.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[21] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[22] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2048–2057, 2015.

[23] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[24] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. *arXiv preprint arXiv:1602.08210*, 2016.